MEMORIA PRÁCTICA 1

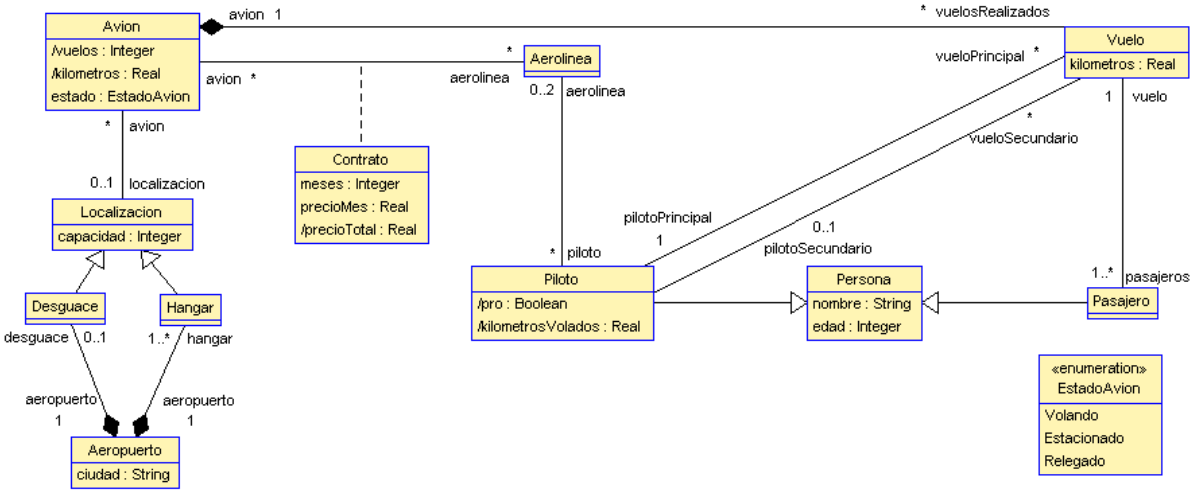
GR2-6

ÍNDICE

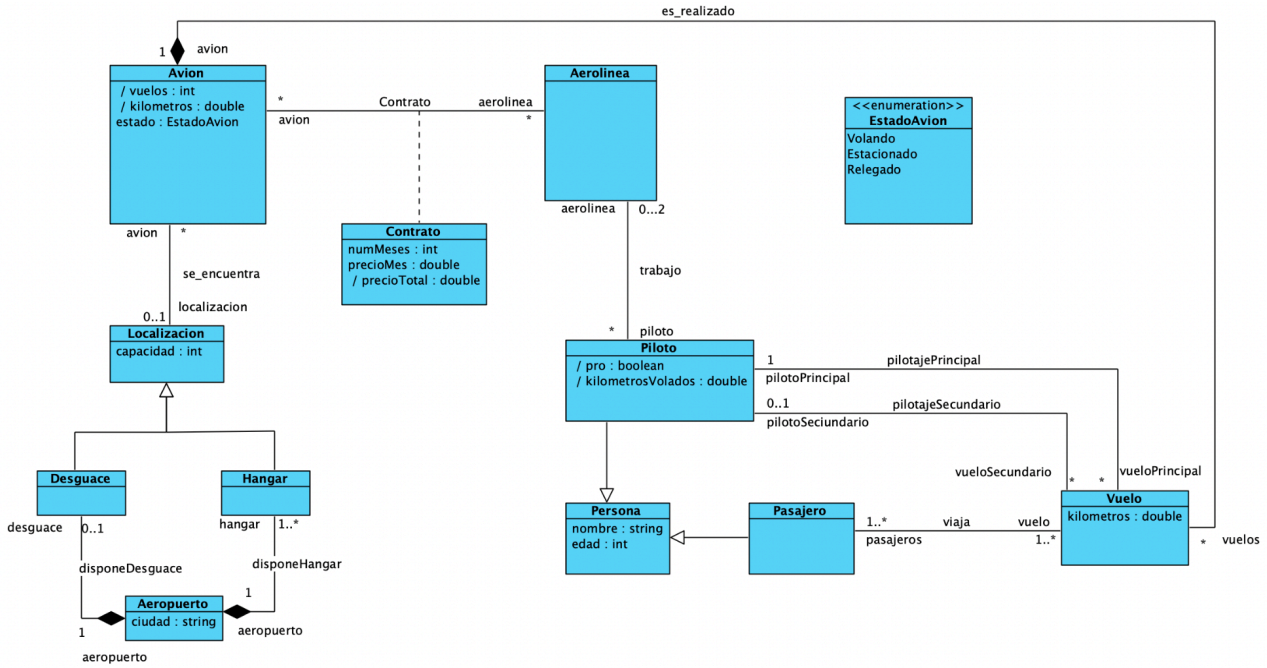
1. Diagramas

1.1. Diagrama de clases

A continuación, se muestra el diagrama de clases obtenido en ***USE***:



A continuación, se muestra el diagrama de clases obtenido en ***Visual Paradigm***:



En el ***diagrama de clases*** expuesto se da solución al problema de aviación planteado. Para ello, contamos con diferentes clases y relaciones que se explicarán a continuación.

En el diagrama podemos identificar las siguientes ***entidades o clases***:

- ***Avión:*** Representa los aviones disponibles en el sistema. Sus atributos son “vuelos” que indica los vuelos asociados al avión, “kilómetros” que refleja el total de kilómetros recorridos y “estado” que indica su estado actual.

- ***Localización:*** Actúa como clase base encapsulando los atributos comunes que comparten todas las localizaciones, en este caso el atributo que tienen en común es “capacidad”.

- ***Desguace:*** Representa los desguaces del sistema, hereda de localización la propiedad de “capacidad”.

- ***Hangar***: Hereda de localización la propiedad de “capacidad”.

- ***Aeropuerto:*** Representa los aeropuertos y en qué ciudad se sitúan los mismos.

- ***Aerolínea:*** Representa las aerolíneas disponibles en el sistema.

- ***Persona:*** Actúa como clase base encapsulando los atributos comunes para la clase “Pasajero” y para la clase “Piloto”. En este caso los atributos que comparten estas clases son “nombre” y “edad”.

- ***Piloto:*** Representa a los pilotos disponibles del sistema. Sus atributos, además de los heredados de la clase persona, son “pro”; un booleano que indica si ya es considerado piloto pro o no y “kilometrosVolados”; que se utiliza para saber cuántos kilómetros ha volado como piloto principal o como secundario.

- ***Pasajero:*** Representa a los pasajeros del sistema. Sus atributos son los heredados de la clase Persona.

- ***Vuelo:*** Representa los vuelos del sistema. Cuenta con un atributo “kilómetros” que indica los kilómetros del trayecto.

- ***EstadoAvión:*** Esta clase es un enumerador utilizado en el sistema para representar los diferentes estados en los que puede estar un vuelo, que son “Volando”, “Estacionado” o “Relegado”.

Por otro lado , podemos destacar la siguiente ***clase de asociación***:

- ***Contrato***: Clase de asociación que representa la relación contractual entre un avión y una aerolínea y cuenta con los atributos “meses”, “precioMes” y “precioTotal”.

Por último, podemos destacar las siguientes ***relaciones*** que permiten asociar las diferentes entidades anteriormente mencionadas:

- La clase “Avión” tiene una relación de asociación con la clase “Localización” llamada “SeEncuentra” que tiene una cardinalidad de “*” en la clase “Avión” y “0…1” en la de “Localizacion”. Lo que indica que un avión se puede encontrar como mucho en una localización, y que en una localización se pueden encontrar desde ningún avión hasta muchos.

- Como ya se explicó anteriormente, “Localizacion” actúa como clase base para “Desguace” y “Hangar” por lo que ambas tienen una relación de herencia con esta clase.

- “Aeropuerto” tiene una relación de composición tanto con “Desguace” como con “Hangar””. En el caso de “Desguace”, la relación se llama “disponeDesguace” y tiene una cardinalidad de “0…1” en la clase “Desguace” y de “1” en la clase “Aeropuerto”, indicando que en un aeropuerto puede haber hasta 1 desguace, y que todo desguace se encuentra en un aeropuerto. En el caso de “Hangar” la relación se llama “disponeHangar” y tiene una cardinalidad de “1…*” en la clase “Hangar” y de “1” en la de Aeropuerto, indicando que un aeropuerto tiene desde un hangar hasta muchos y que un hangar se encuentra en un aeropuerto.

- “Avión” y “Aerolinea” tienen una relación de asociación y a su vez cuentan con la clase de asociación “Contrato”. La cardinalidad de esa asociación es de “*” tanto por parte de “Avión” como de “Aerolinea”, indicando que un avión puede estar en tantas aerolíneas como quiera y que en una aerolínea puede haber tantos aviones como se necesite.

- “Persona” sirve de clase base para “Piloto” y para “Pasajero” por lo que tienen una relación de herencia con ella.

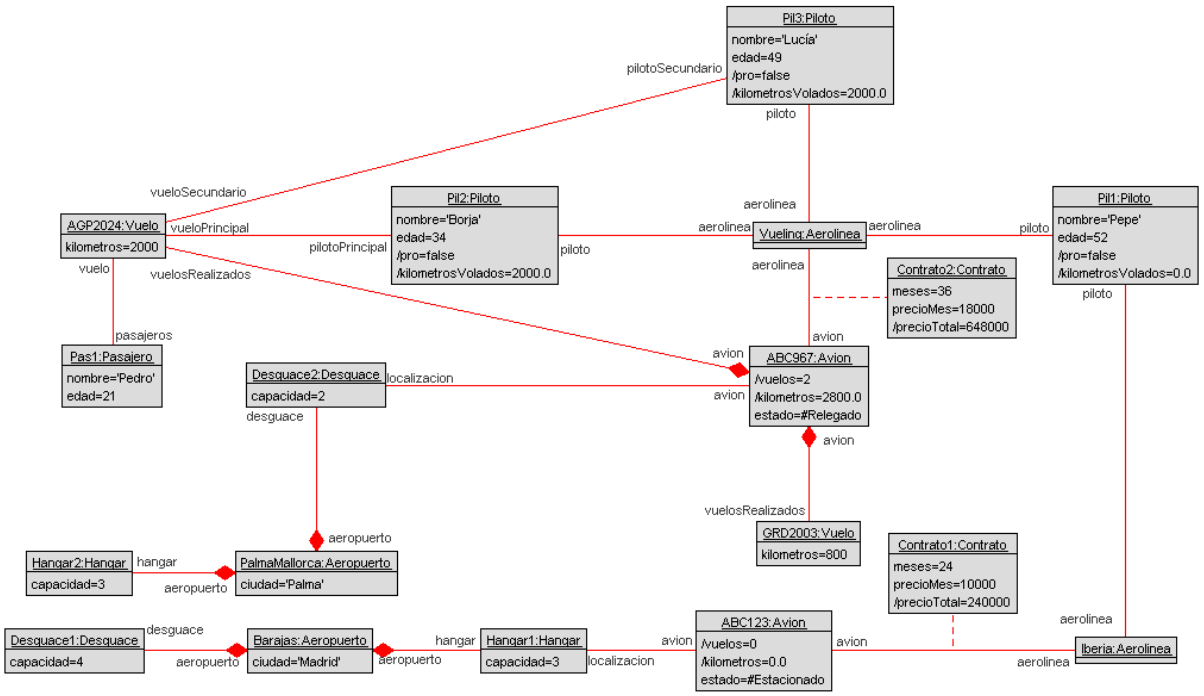
- La clase “Piloto” tiene dos relaciones de asociación con la clase “Vuelo” llamadas “PilotajePrincipal” y “PilotajeSecundario”. “PilotajePrincipal” tiene una cardinalidad de “1” y “*” en la clase “Piloto” y “Vuelo” respectivamente, indicando así que un Piloto puede ser el piloto principal de muchos vuelos mientras que cada Vuelo solo puede tener un piloto principal. “PilotajeSecundario” tiene una cardinalidad de “0..1” y “*” en la clase “Piloto” y “Vuelo” respectivamente, indicando que puede no haber Piloto Secundario y de haberlo puede asistir a muchos vuelos. A su vez piloto tiene una relación llamada “trabajo” con “Aerolinea” que tiene una cardinalidad de “*” y de “0…2” respectivamente, lo que indica que un piloto puede trabajar como mucho para dos aerolíneas y que una aerolínea puede tener tantos pilotos como necesite.

- “Pasajero” y “Vuelo” tienen una relación de asociación llamada “Viaja” con una cardinalidad de “1…*” y “1”, lo que indica que un pasajero puede viajar en un vuelo y que un vuelo puede tener muchos pasajeros.

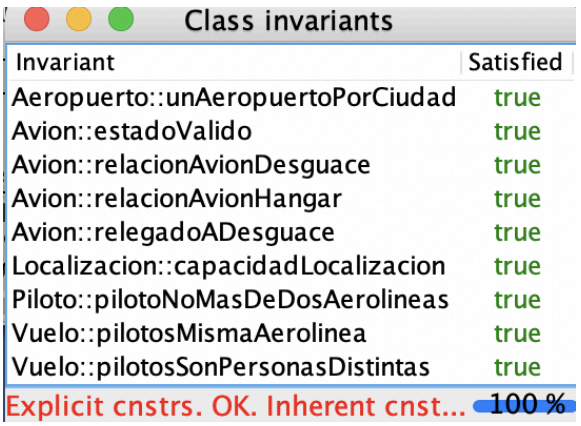
1.2. Diagramas de objetos

1.2.1. Diagrama Aviacion.soil

El diagrama de objetos **Aviacion.soil** , es aquel que hemos usado para demostrar que todas las invariantes se satisfacen, es decir, que no violan ningúna restricción.



En este caso, podemos observar que todas las invariantes se muestran a **'true'**:



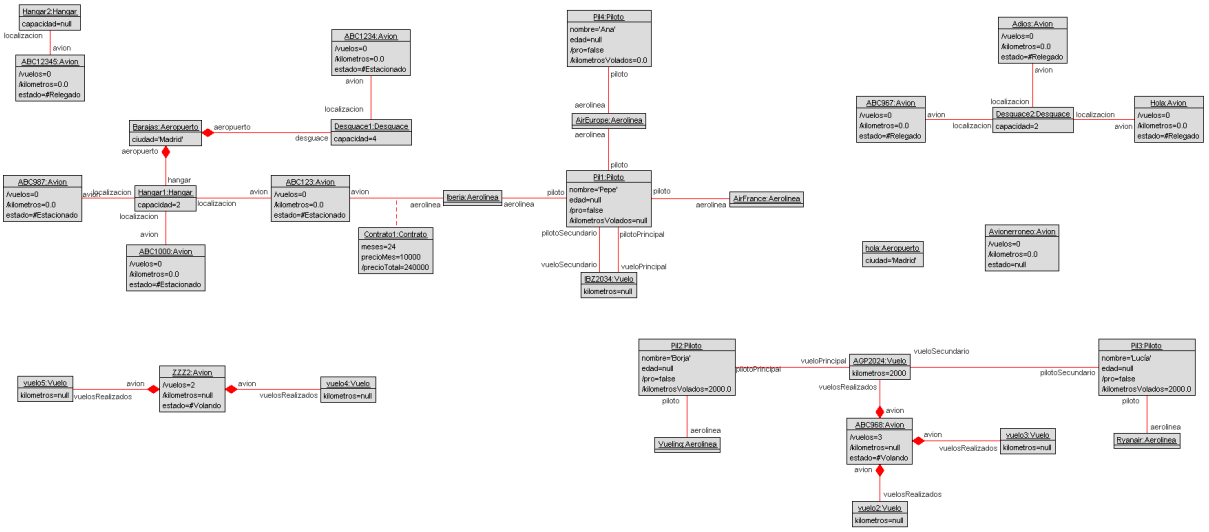
En este caso, hemos creado objetos que verifiquen las invariantes positivamente.

Cada una de las invariantes asegura una condición específica:

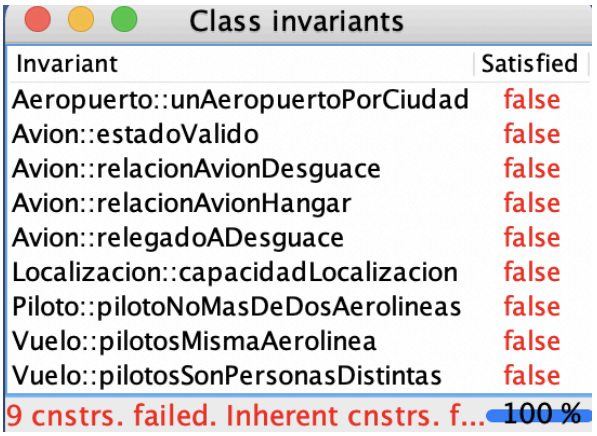
- pilotoNoMasDeDosAerolineas: Esta invariante garantiza que ningún piloto trabaje para más de dos aerolíneas de manera simultánea. En el ejemplo, “Pil2” y “Pil3” trabajan para la aerolínea Vueling, mientras que “Pil1” está asociado tanto a Vueling como a Iberia. No hay pilotos trabajando para más de dos aerolíneas por lo que se verifica la condición.
- unAeropuertoPorCiudad: Asegura que sólo haya un aeropuerto por ciudad. En el ejemplo, en la ciudad de Palma hay un único aeropuerto llamado “PalmaMallorca”, lo cual satisface la invariante, ocurre lo mismo con el aeropuerto “Barajas” que sólo está en la ciudad de Madrid.
- estadoValido: Verifica que el estado de cada avión sea “Volando”, “Estacionado” o “Relegado”. En el ejemplo, el avión “ABC123” está en estado “Estacionado” mientras que el “ABC967” está en “Relegado”. Ambos estados son válidos y se verifica la condición.
- relacionAvionDesguace: Está relación verifica que los aviones cuyo estado sea “Relegado” se encuentren en el desguace. Se verifica ya que el avión “ABC967” que está relegado está asociado a un desgüace.
- relacionAvionHangar: Esta relación verifica que los aviones cuyo estado sea “Estacionado” se encuentre en el hangar. Se verifica ya que el avión “ABC123” que está estacionado está asociado a un hangar.
- relegadoDesguace: Esta invariante verifica que un avión es relegado a un desguace una vez complete los 1000 vuelos. Para comprobarlo cambiamos el valor a 2 vuelos y como se puede observar el avión pasa a ser relegado al desguace.
- capacidadLocalizacion: La capacidad de los hangares y desguaces también se verifica ya que el “Hangar1” tiene una capacidad de 3 y sólo cuenta con un avión así como el “Desguace2” tiene una capacidad de 2 aviones y cuenta con uno.
- pilotoMismaAerolinea: En esta invariante se verifica que los vuelos tengan pilotos de la misma aerolínea. Se verifica ya que en el vuelo “AGP2024” tiene a “Pil2” y a “Pil3” como pilotos, siendo ambos pilotos de la aerolínea Vueling.
- pilotosSonPersonasDistintas: Esta invariante verifica que cuando haya un piloto principal y un piloto secundario sean personas distintas. Al estar “Pil2” y “Pil3” en el vuelo AGP2024, se verifica esta condición.

1.2.2. Diagrama ErrorAviacion.soil

El diagrama de objetos **ErrorAviacion.soil**, es el que hemos usado para comprobar que las invariantes no se satisfacen en los casos especificados en el enunciado de la práctica.



En este caso, podemos observar que ninguna invariante se satisface:



En este caso, hemos creado objetos que verifiquen las invariantes negativamente:

- pilotoNoMasDeDosAerolinea: En este ejemplo, hemos asignado a “Pil1” a AirFrance, AirEurope y a Iberia, por lo que la invariante no se cumple.
- unAeropuertoPorCiudad: En el ejemplo se puede observar que ahora la ciudad de Madrid cuenta con dos aeropuertos, “Barajas” y “Hola” por lo que la invariante no se cumple.

- estadoValido: La invariante no se cumple ya que el avión “ABC967”, “Avión” y “Hola” no tiene un estado coherente a su localización ya que están en estado relegado sin haber hecho ningún vuelo.
- relacionAvionDesguace: La invariante no se cumple debido al ejemplo anterior ya que los tres aviones no tienen un estado coherente a su localización.
- relacionAvionHangar: La invariante no se cumple ya que el avión “ABC12345” se encuentra en estado relegado en un hangar.
- relegadoADesguace: La invariante no se cumple ya que el avión “ABC968” ha hecho 3 vuelos por lo que debería estar relegado (ya que la condición ha sido cambiada para testear) pero se encuentra volando.
- capacidadLocalizacion: La invariante no se cumple ya que el avión “ABC12345” se encuentra en un hangar que tiene una capacidad nula.
- pilotosMismaAerolinea: La invariante no se cumple ya que en el vuelo AGP2024, “Pil2” y “Pil3” son pilotos a pesar de trabajar en compañías distintas.
- pilotosSonPersonasDistintas: Esta invariante no se cumple ya que en el vuelo “IBZ2034”, “Pil1” es piloto principal y piloto secundario, por lo que no hay pilotos distintos.

2. Invariantes o restricciones OCL

A continuación, se muestran las diferentes ***invariantes*** que hemos establecido:

```
--RESTRICCIONES-----------------------------------------------------------------------------------------------
constraints

context Avion
    -- Todo avión debe encontrarse volando, estacionado en un hangar o ha sido ya relegado.
    inv estadoValido: self.estado = #Volando or self.estado = #Estacionado or self.estado = #Relegado

    -- Los aviones que deben encontrarse en un desguace son aquellos que han completado los 1000 viajes.
    inv relegadoADesguace: self.vuelos >= 1000 implies self.estado = #Relegado

    -- Existe relacion entre avión y hangar cuando el estado del avión es estacionado.
    inv relacionAvionHangar: self.estado = #Estacionado implies self.localizacion.oclIsTypeOf(Hangar)

    -- Existe relacion entre avión y desguace cuando el estado del avión es relegado.
    inv relacionAvionDesguace: self.estado = #Relegado implies self.localizacion.oclIsTypeOf(Desguace)

context Vuelo
    -- Lógicamente, los pilotos principal y secundario deben ser personas distintas.
    inv pilotosSonPersonasDistintas: if self.pilotoSecundario <> null then self.pilotoPrincipal <> self.pilotoSecundario else true endif

    -- Piloto principal y secundario trabajan en la misma aerolínea
    inv pilotosMismaAerolinea: self.pilotoSecundario <> null implies self.pilotoPrincipal.aerolinea = self.pilotoSecundario.aerolinea

context Aeropuerto
    -- Los aeropuertos se ubican en ciudades, no pudiendo haber más de un aeropuerto en cada ciudad.
    inv unAeropuertoPorCiudad: Aeropuerto.allInstances() -> isUnique(a | a.ciudad)

context Piloto
    -- No se permite a un piloto trabajar o haber trabajado para más de dos aerolíneas diferentes. Cogemos todas las aerolineas donde trabaja piloto.
    inv pilotoNoMasDeDosAerolineas: self.aerolinea -> size() <= 2

context Localizacion
    -- La cantidad de aviones en la localización no debe exceder su capacidad.
    inv capacidadLocalizacion: self.avion -> size() <= self.capacidad
```

3. Consideraciones adicionales

Para modelar la ciudad hemos considerado también crear una clase Ciudad y establecer una relación de 1 a 1 con la clase Aeropuerto. Sin embargo, hemos optado por representarla como un atributo de tipo String y añadir la restricción unAeropuertoPorCiudad ya que no hemos encontrado diferencias prácticas para este caso concreto y estimamos que ambas soluciones pueden ser igualmente válidas.

La restricción estadoValido se ha añadido para asegurarnos de que no hay ningún error en el estado asignado a los aviones ya que, cuando el modelo de objetos es de un tamaño considerable, es más difícil detectar un error al cargar el modelo de objetos o ver la asignación a null.